ECE 6882 – Reinforcement Learning

Spring 2023 – The George Washington University

Osama Yousuf – osamayousuf@gwu.edu

Project 2 – Solving a Maze using Policy Iteration & Value Iteration

In this project, I have implemented two different policies to solve the provided maze problem, and I have carried out experiments over various algorithmic hyperparameters to establish a sense of which algorithm performs better than others. I used environment 4 and edited its functionality to my liking. Supplemental code files are provided as an accompanying archive. Table 1 highlights the purpose of each file.

Table 1: Description of implemented code.

File	Description
main.py	Entry point. Executing this as is will replicate all plots used in this report.
policies.py	Contains my implementations of random policy, policy iteration, and value iteration.
util.py	Contains environment-specific functionality such as classes for the maze and agent, as well as visualization functions.
mazes/base.txt	This is the base file for the 18 x 18 maze which we have to solve for this project. This is directly used in main.py. This could be replaced with any other maze directly, and the environment should correctly load and solve it using both policy iteration as well value iteration.

As highlighted in Table 1, my environment can load a maze for solving using generic text files, as long as the mapping is respected. Figure 1 shows the base maze, and Table 2 highlights the reward values associated with each state in the maze.

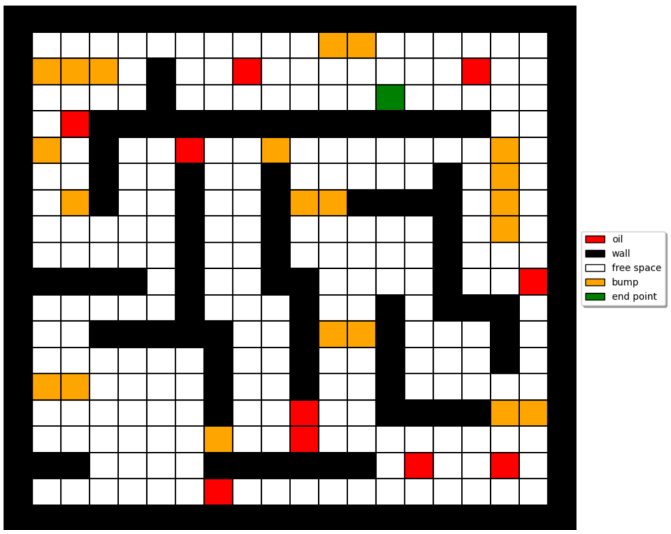

Figure 1: Base maze environment loaded from mazes/base.txt

State (after transition)	Reward
All actions (Up, Down, Left, Right)	-1
Oil	-5
Bump	-10
Goal	+200

Table 2: Reward as a function of transitioned state

Firstly, the agent receives a -1 reward for all actions regardless of the state it transitions to. Secondly, the agent receives additional special rewards if it transitions to oil, bump, and goal states, as specified in Table 2.

Experimental Scenarios

There are three scenarios based on the hyperparameters for the two policies which I have explored. These are:

1) Base Scenario: $p = 0.02$, $\gamma = 0.95$, $\theta = 0.01$
2) High Stochasticity Scenario: $p = 0.5$, $\gamma = 0.95$, $\theta = 0.01$
3) Small Discount Factor Scenario: $p = 0.02$, $\gamma = 0.55$, $\theta = 0.01$

Parameter p refers to the transition randomness of the agent, since the agent transitions to an intended state with a probability of 1 – p. Higher the value of p (between 0 and 1, inclusive), higher is the randomness in the agent in following an intended trajectory.

Parameter γ refers to the discount factor alotted to future rewards. It directly represents the importance of future rewards relative to immediate rewards. Higher the value (between 0 and 1, inclusive), the more important future rewards are considered relative to the immediate reward.

Finally, parameter θ is simply a small positive number which determines the accuracy of estimation of the optimal policy under policy iteration as well as value iteration. Its value can be made smaller to find a better policy, tradeoff being a higher convergence time.

Policy Iteration

Base Scenario

Figures 2, 3, 4 report the optimal policy, V function values, and the optimal path respectively learned by policy iteration under this scenario. In addition, Figure 5 shows an instance of an agent attempting to traverse the maze using the policy with transition randomness in place.

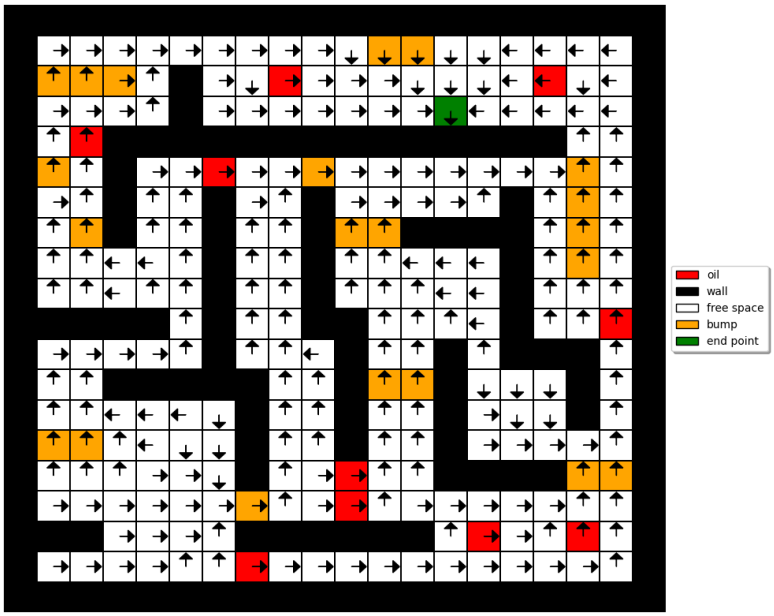

Figure 2: Optimal policy learned by Policy Iteration under the base scenario

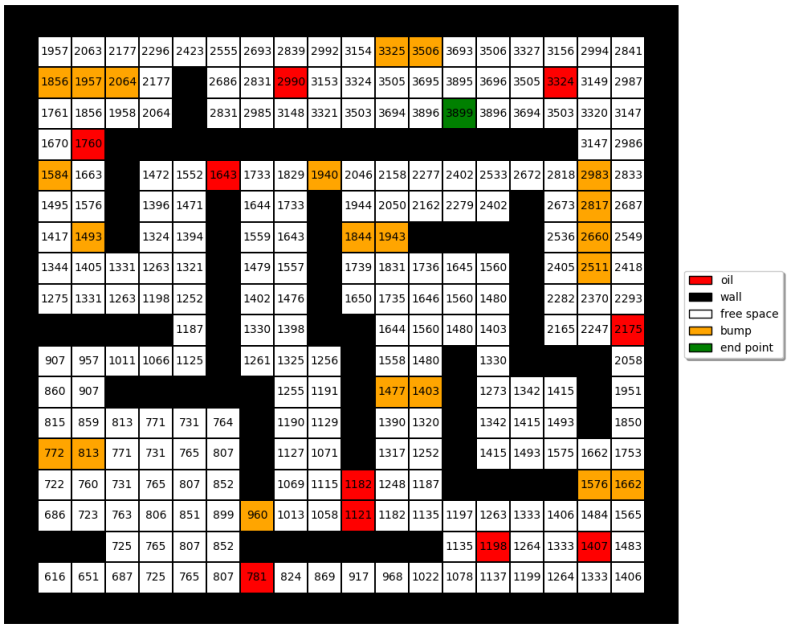

Figure 3: Optimal V function learned by Policy Iteration under the base scenario

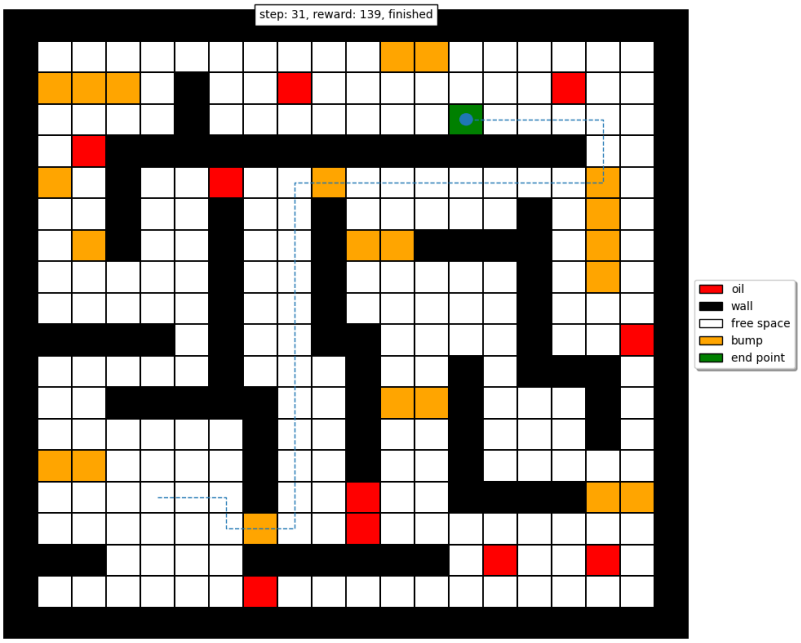

Figure 4: Optimal path learned by Policy Iteration under the base scenario

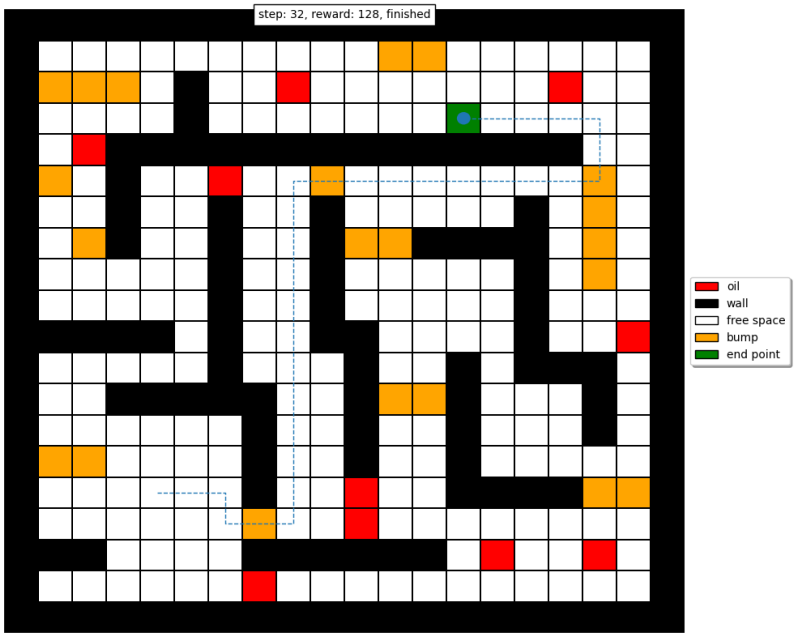

Figure 5: Instance of an agent solving the maze under the given scenario and learned policy

It can be seen that the trajectories in Figures 4 and 5 are similar, with only a single extra step as the agent is quickly able to recover under the learned policy. This occurs because we have really small transition randomness ($p = 0.02$), so the agent transitions to an intended state 98% of the time.

The large V function values in Figure 3 can be explained similarly – once the agent reaches the goal state, it can take the optimal action of down, hit a wall, remain in the goal state, and repeatedly receive the goal

state reward of +200. The expected reward of each state thus accumulates overtime and increases in magnitude, as seen in the V function values.

High Stochasticity Scenario

Figures 6, 7, 8 report the optimal policy, V function values, and the optimal path respectively learned by policy iteration under this scenario. In addition, Figure 9 shows an instance of an agent attempting to traverse the maze using the policy with transition randomness in place.

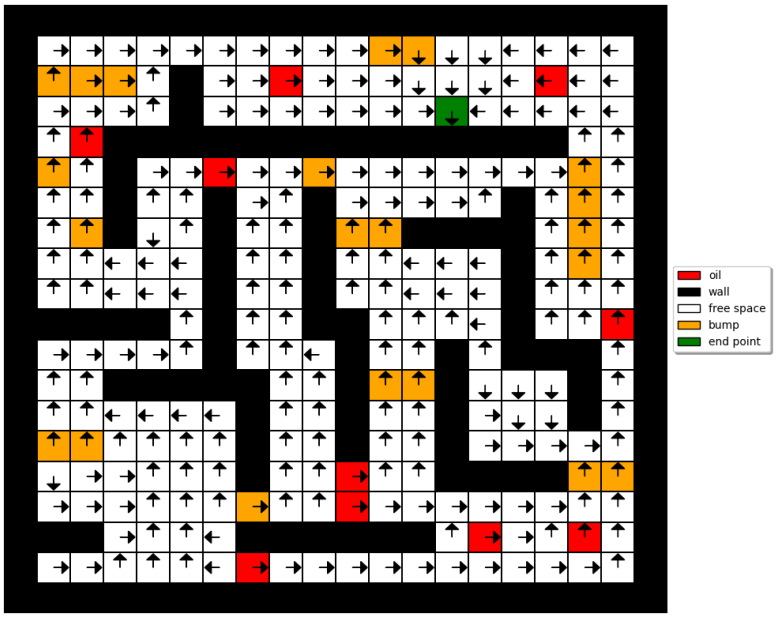

Figure 6: Optimal policy learned by Policy Iteration under the base scenario

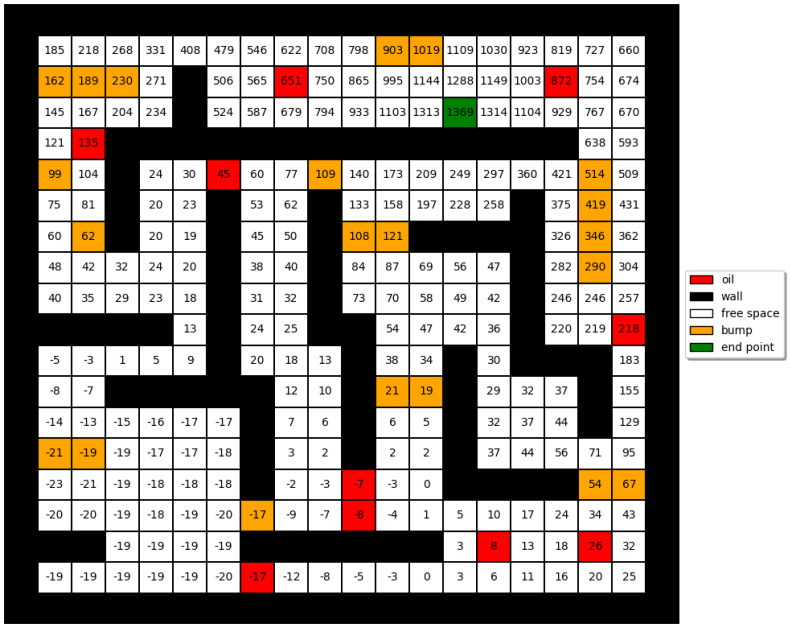

Figure 7: Optimal V function learned by Policy Iteration under the base scenario

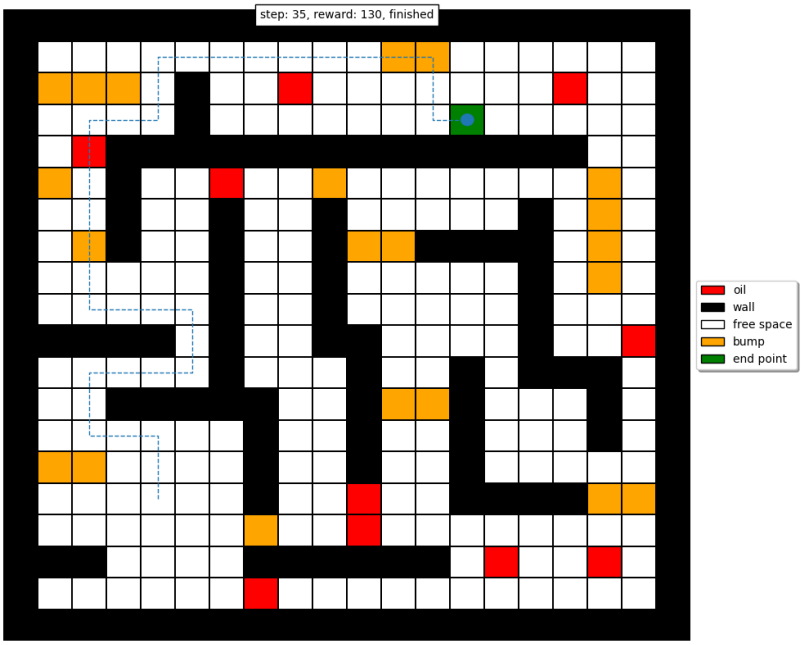

Figure 8: Optimal path learned by Policy Iteration under the base scenario

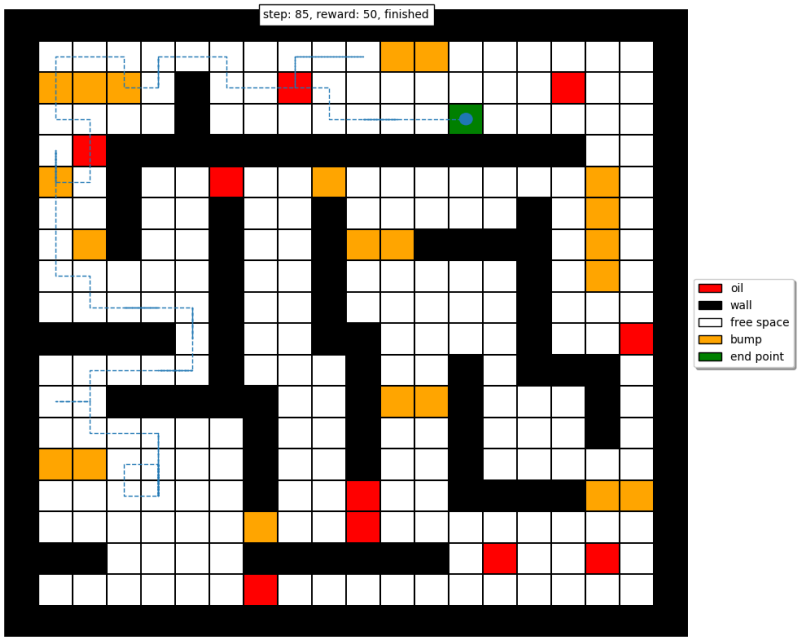

Figure 9: Instance of an agent solving the maze under the given scenario and learned policy

It can be seen that the trajectories in Figures 8 and 9 have a high variation, indicating that the transition randomness is too high for the agent to be able to quickly recover under the learned policy. This occurs because we have a relatively high transition randomness ($p = 0.5$), so the agent transitions to an intended state only 50% of the time and wanders randomly in an unintended direction the remaining 50% of the time.

Compared to the V function values in the base scenario, the V function values in Figure 7 are much smaller. This can be explained along similar lines as the base scenario – once the agent reaches the goal state, it can

take the optimal action with only a probability of 50%, and so there's always a high chance of transitioning to a state that is not the goal state, thereby reducing the expected accumulated reward.

Finally, it can be seen that under this scenario, the optimal path that is learned is sub-optimal compared to the path which is learned in the base scenario. This can be attributed to the fact that the transition randomness led to a much higher degree of exploration, and the agent was unable to arrive at the same optimal policy. It is possible that other algorithmic parameters such as the discount factor and accuracy of estimation could be altered to learn a much better policy overall.

Small Discount Factor Scenario

Figures 10, 11, 12 report the optimal policy, V function values, and the optimal path respectively learned by policy iteration under this scenario. In addition, Figure 13 shows an instance of an agent attempting to traverse the maze using the policy with transition randomness in place.

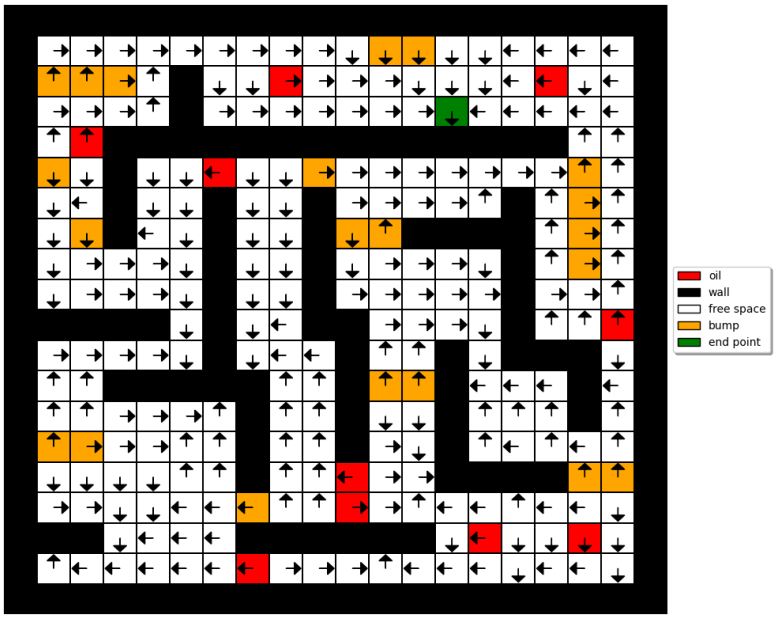

Figure 10: Optimal policy learned by Policy Iteration under the base scenario

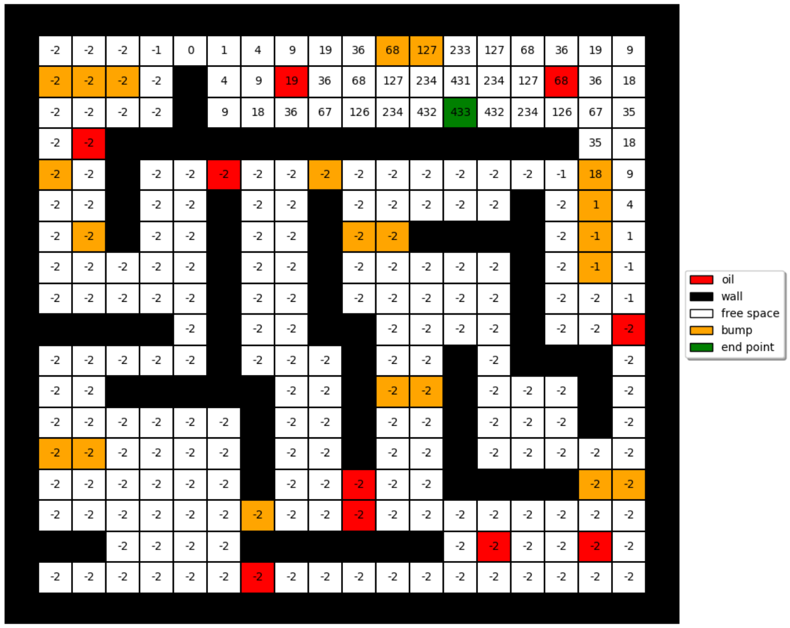

Figure 11: Optimal V function learned by Policy Iteration under the base scenario

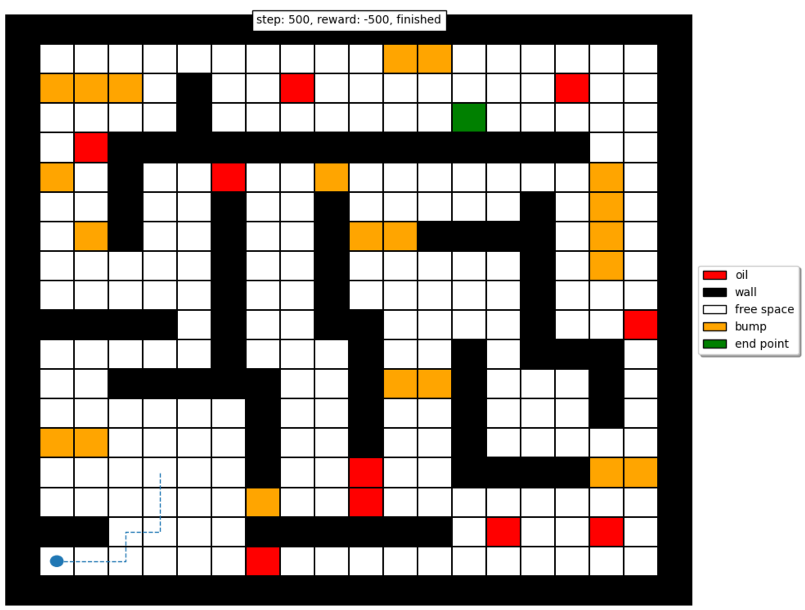

Figure 12: Optimal path learned by Policy Iteration under the base scenario

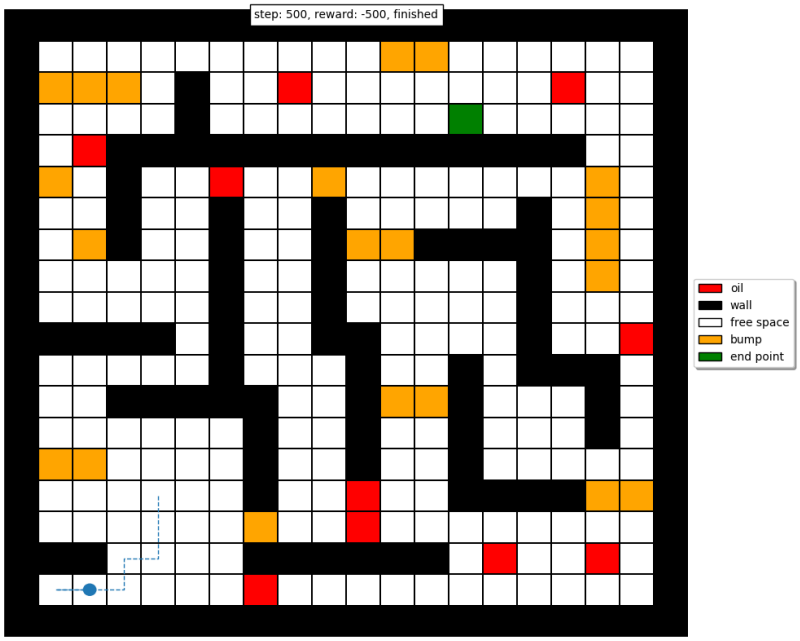

Figure 13: Instance of an agent solving the maze under the given scenario and learned policy

It can be seen that the trajectories in Figures 12 and 13 have only a slight variation, but both are equally bad since the agent is unable to get to the goal state from the starting position. Since we have small transition randomness (2%), the agent remains trapped in the corner as it follows the bad policy shown in Figure 10.

Compared to the V function values in the previous two scenarios, the V function values in Figure 11 are the smallest. In addition, the optimal path that is learned is entirely sub-optimal since the agent remains trapped under the learned policy. Both of these things can be attributed to the smaller discount factor. A higher discount factor encourages the agent to optimize for long-term rewards, while a lower discount factor focuses on immediate rewards. From the V values, it can be seen that the agent has hyper-optimized for immediate rewards, leading to a policy that basically traps the agent if the starting position is sufficiently far from the goal states. From our results in the base scenario, we already know that by increasing the discount factor from 0.55 to 0.95, we can learn a really good optimal policy.

Table 3 offers a comparison of the performance of policy iteration under the three investigated scenarios. It can be seen that the base scenario is the optimal choice since the agent is able to achieve an optimal policy that guarantees the highest reward while minimizing the number of steps taken to get to the goal state. The small discount factor scenario exhibits the worst performance, since the policy fails to solve the maze altogether under the given starting position.

Table 3: Performance comparison of Policy Iteration under the three investigated scenarios

Scenario	Steps	Final Reward
Base	31	139
High Stochasticity	35	130
Small Discount Factor	500	-500 (agent stuck, maze unsolved)

Value Iteration

Base Scenario

Figures 14, 15, 16 report the optimal policy, V function values, and the optimal path respectively learned by value iteration under this scenario. In addition, Figure 17 shows an instance of an agent attempting to traverse the maze using the policy with transition randomness in place.

In addition, it can be seen that the learned optimal policy, V function, and path are the same as the ones learned by policy iteration.

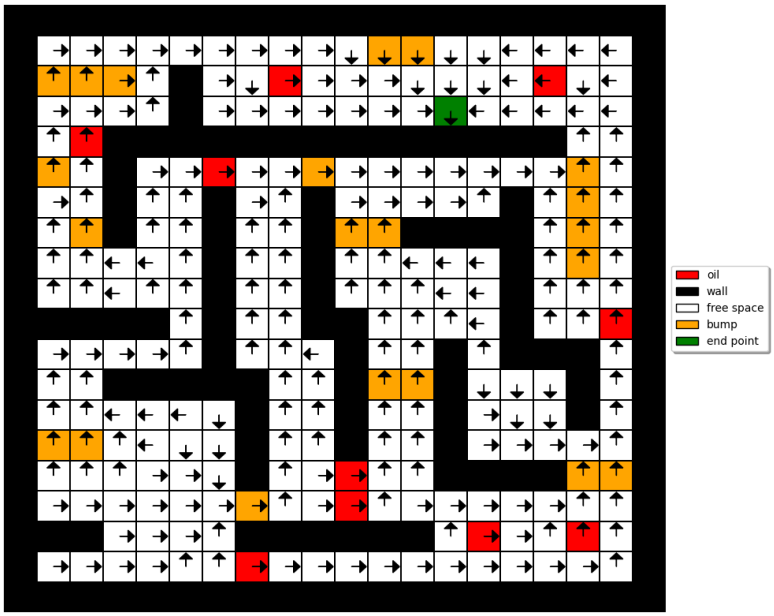

Figure 14: Optimal policy learned by Policy Iteration under the base scenario

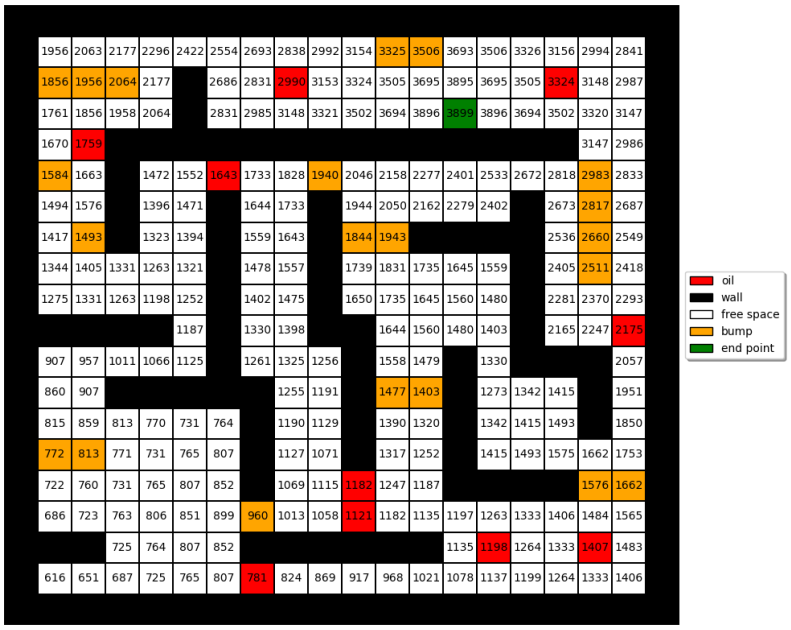

Figure 15: Optimal V function learned by Policy Iteration under the base scenario

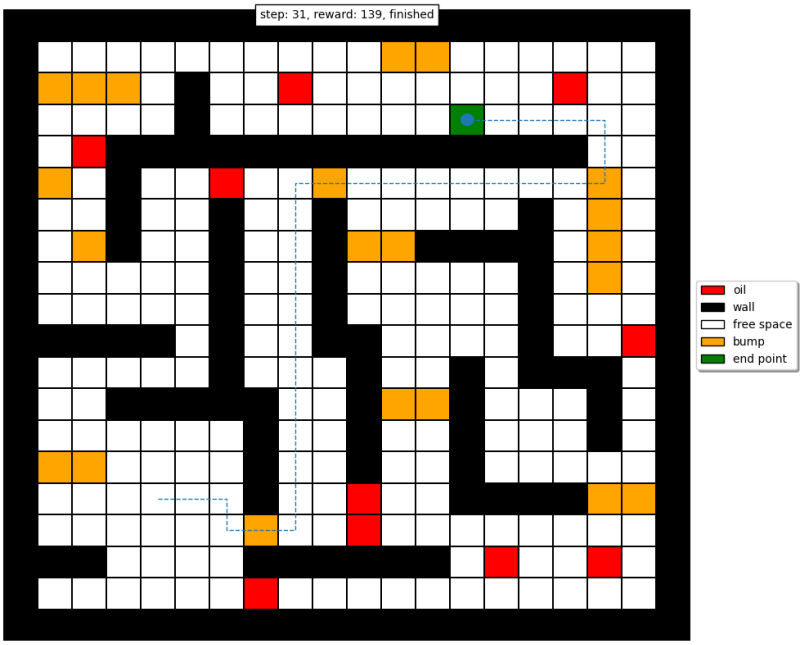

Figure 16: Optimal path learned by Policy Iteration under the base scenario

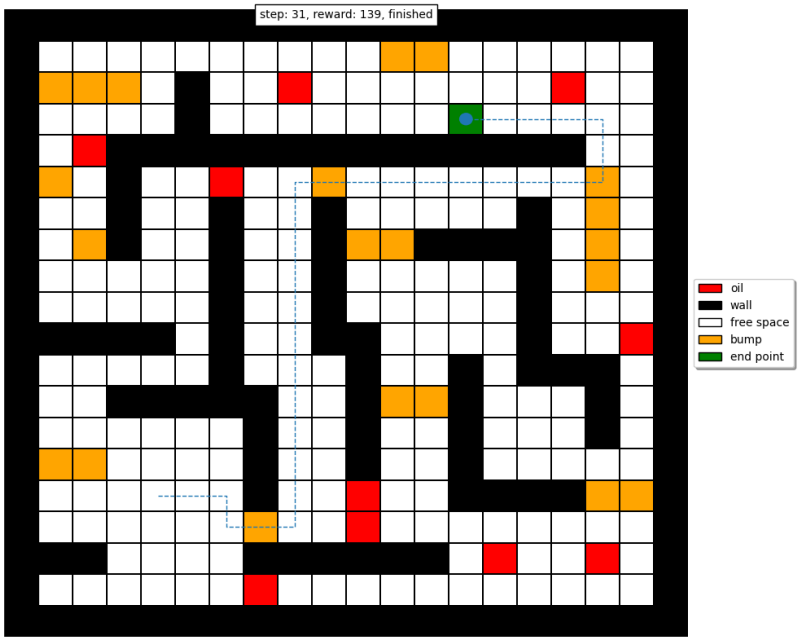

Figure 17: Instance of an agent solving the maze under the given scenario and learned policy

It can be seen that the trajectories in Figures 16 and 17 are similar. This occurs because we have really small transition randomness (p = 0.02), so the agent transitions to an intended state 98% of the time.

The large V function values in Figure 15 can be explained similarly – once the agent reaches the goal state, it can take the optimal action of down, hit a wall, remain in the goal state, and repeatedly receive the goal state reward of +200. The expected reward of each state thus accumulates overtime and increases in magnitude, as seen in the V function values.

High Stochasticity Scenario

Figures 18, 19, 20 report the optimal policy, V function values, and the optimal path respectively learned by policy iteration under this scenario. In addition, Figure 21 shows an instance of an agent attempting to traverse the maze using the policy with transition randomness in place.

In addition, it can be seen that the learned optimal policy, V function, and path are the same as the ones learned by policy iteration.

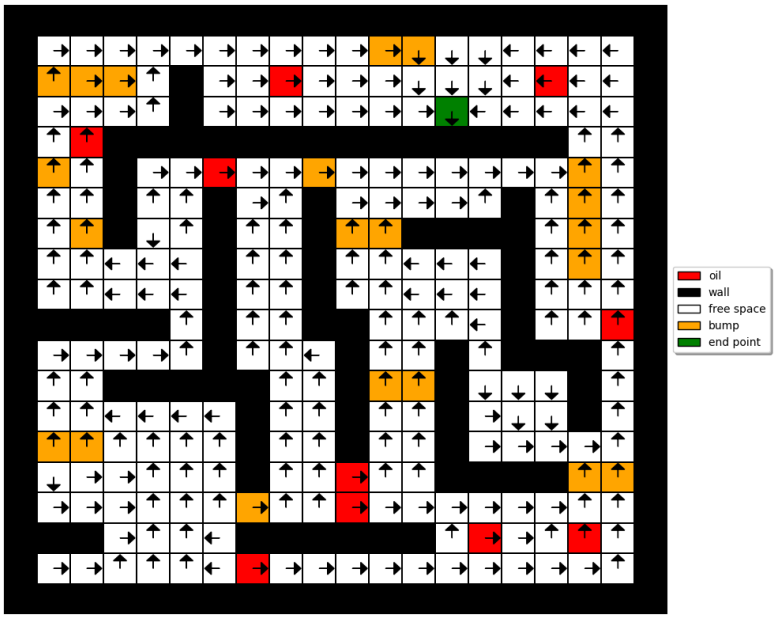

Figure 18: Optimal policy learned by Policy Iteration under the base scenario

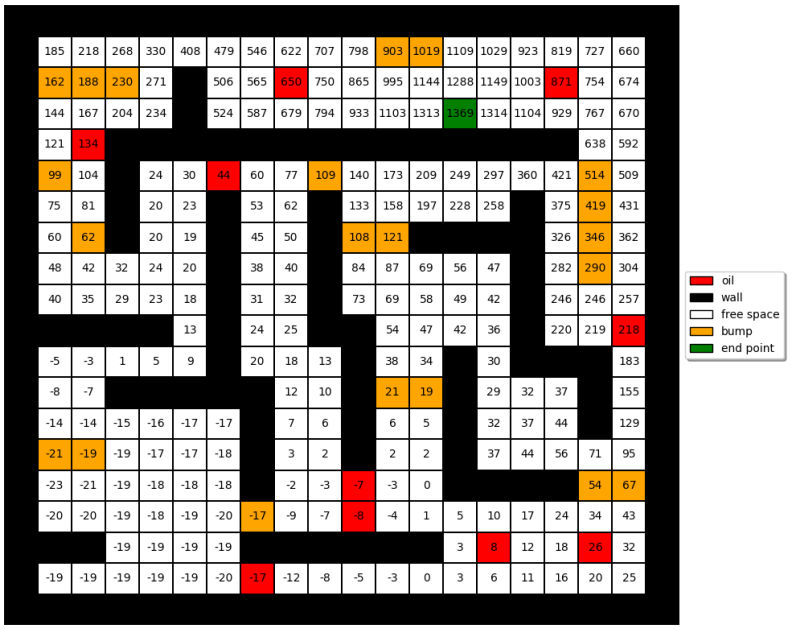

Figure 19: Optimal V function learned by Policy Iteration under the base scenario

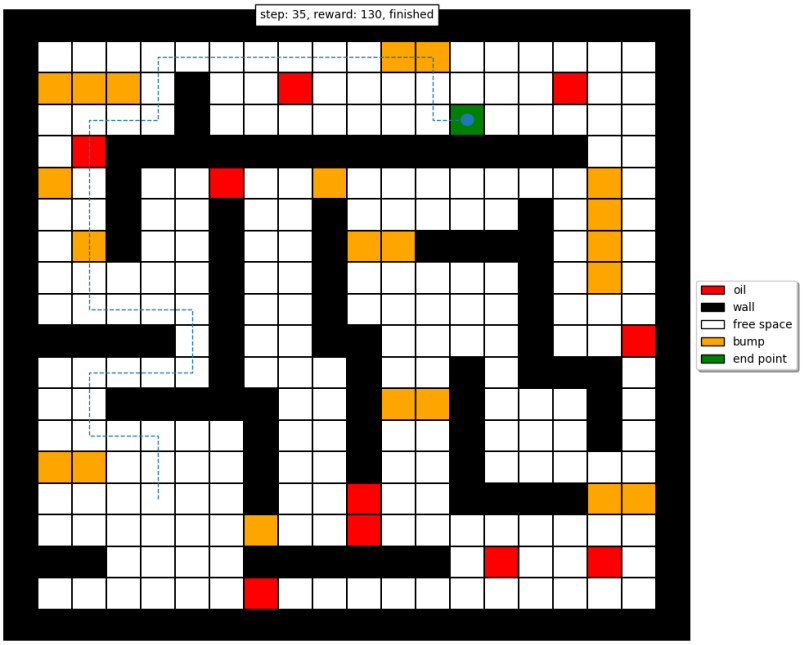

Figure 20: Optimal path learned by Policy Iteration under the base scenario

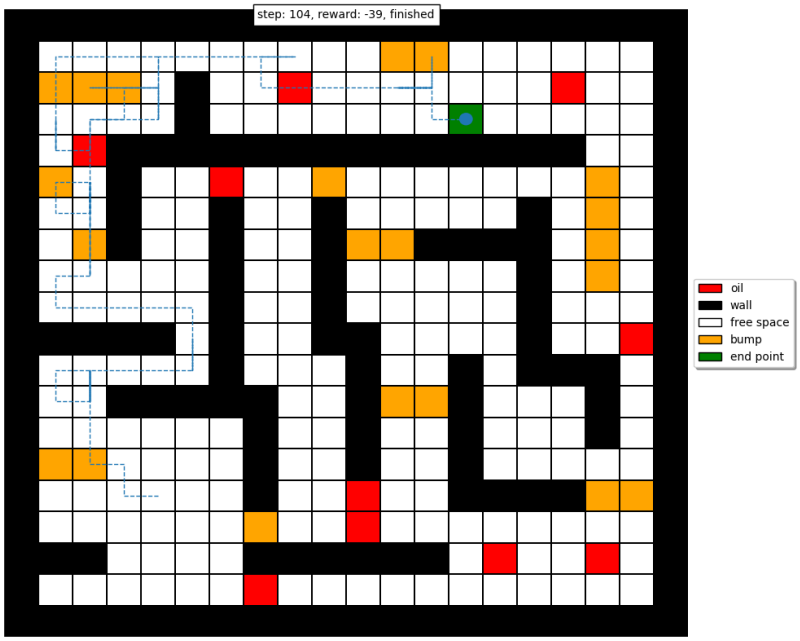

Figure 21: Instance of an agent solving the maze under the given scenario and learned policy

It can be seen that the trajectories in Figures 20 and 21 have a high variation, indicating that the transition randomness is too high for the agent to be able to quickly recover under the learned policy. This occurs because we have a relatively high transition randomness ($p = 0.5$), so the agent transitions to an intended state only 50% of the time and wanders randomly in an unintended direction the remaining 50% of the time.

Compared to the V function values in the base scenario, the V function values in Figure 19 are much smaller. This can be explained along similar lines as the base scenario – once the agent reaches the goal

state, it can take the optimal action with only a probability of 50%, and so there's always a high chance of transitioning to a state that is not the goal state, thereby reducing the expected accumulated reward.

Finally, it can be seen that under this scenario, the optimal path that is learned is sub-optimal compared to the path which is learned in the base scenario. This can be attributed to the fact that the transition randomness led to a much higher degree of exploration, and the agent was unable to arrive at the same optimal policy. It is possible that other algorithmic parameters such as the discount factor and accuracy of estimation could be altered to learn a much better policy overall.

Small Discount Factor Scenario

Figures 22, 23, 24 report the optimal policy, V function values, and the optimal path respectively learned by policy iteration under this scenario. In addition, Figure 25 shows an instance of an agent attempting to traverse the maze using the policy with transition randomness in place.

In addition, it can be seen that the learned optimal policy, V function, and path are similar to the ones learned by policy iteration.

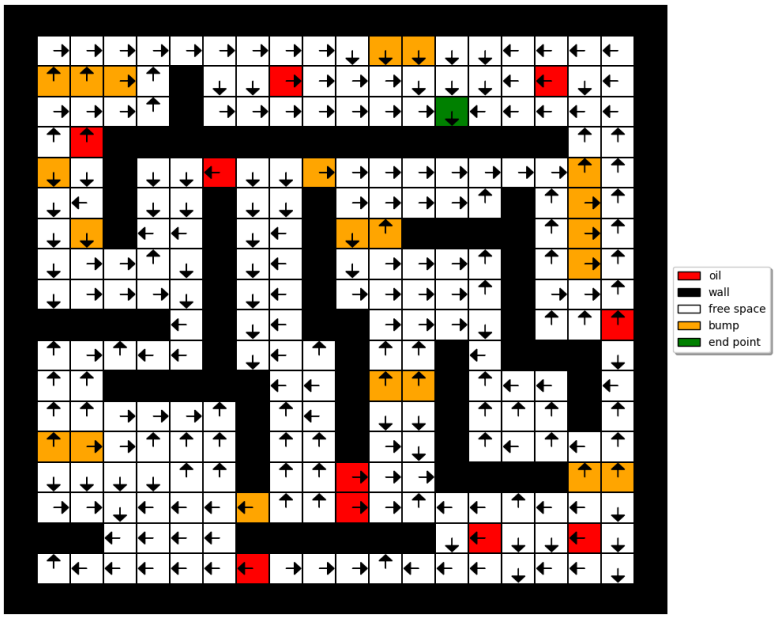

Figure 22: Optimal policy learned by Policy Iteration under the base scenario

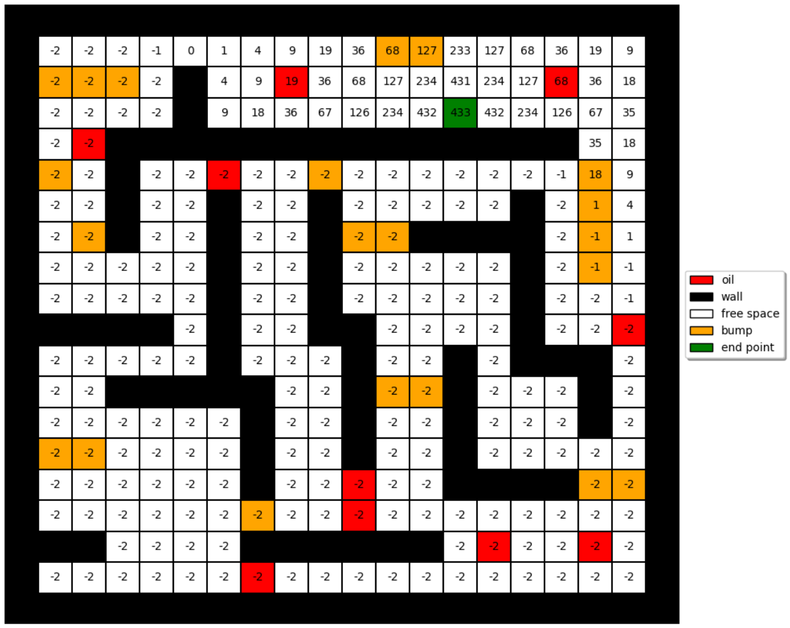

Figure 23: Optimal V function learned by Policy Iteration under the base scenario

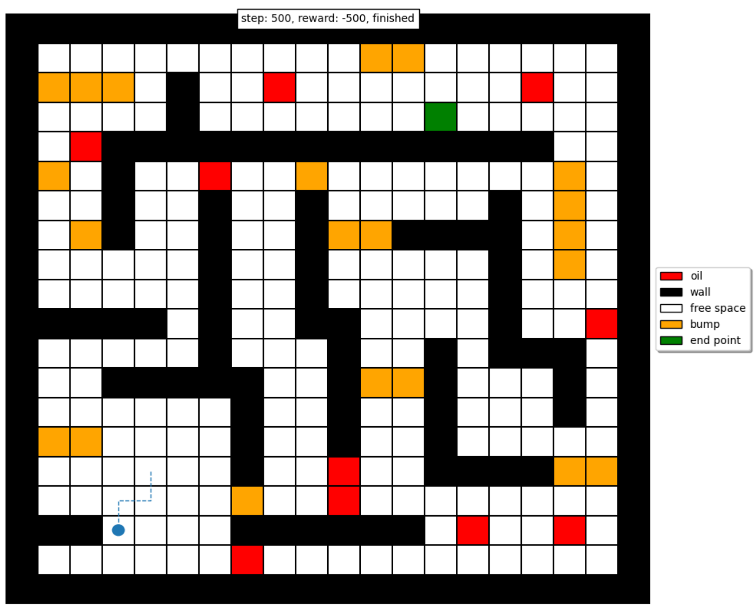

Figure 24: Optimal path learned by Policy Iteration under the base scenario

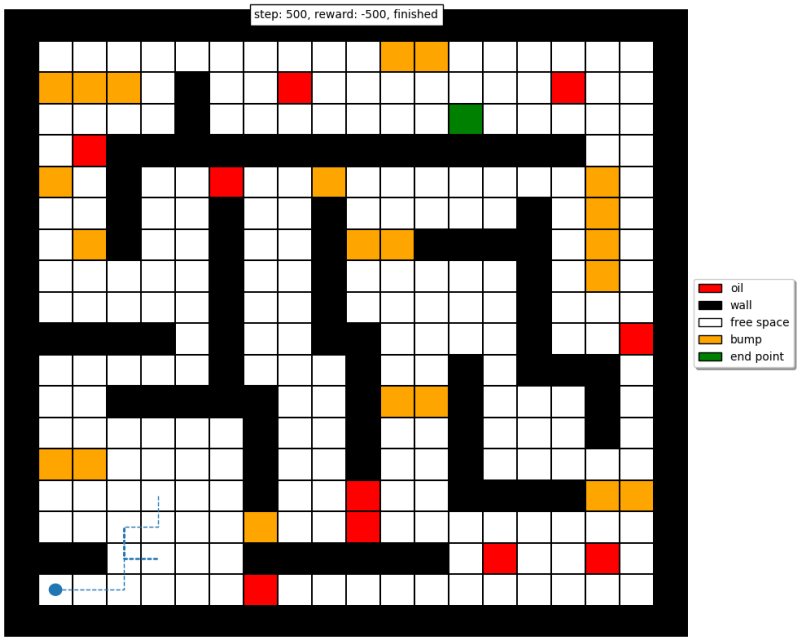

Figure 25: Instance of an agent solving the maze under the given scenario and learned policy

It can be seen that the trajectories in Figures 24 and 25 have only a slight variation, but both are equally bad since the agent is unable to get to the goal state from the starting position. Since we have small transition randomness (2%), the agent remains trapped in the corner as it follows the bad policy shown in Figure 22.

Compared to the V function values in the previous two scenarios, the V function values in Figure 23 are the smallest. In addition, the optimal path that is learned is entirely sub-optimal since the agent remains trapped under the learned policy. Both of these things can be attributed to the smaller discount factor. A higher discount factor encourages the agent to optimize for long-term rewards, while a lower discount factor focuses on immediate rewards. From the V values, it can be seen that the agent has hyper-optimized for immediate rewards, leading to a policy that basically traps the agent if the starting position is sufficiently far from the goal states. From our results in the base scenario, we already know that by increasing the discount factor from 0.55 to 0.95, we can learn a really good optimal policy.

Table 3 offers a comparison of the performance of policy iteration under the three investigated scenarios. It can be seen that the base scenario is the optimal choice since the agent is able to achieve an optimal policy that guarantees the highest reward while minimizing the number of steps taken to get to the goal state. The small discount factor scenario exhibits the worst performance, since the policy fails to solve the maze altogether under the given starting position.

Table 4: Performance comparison of Value Iteration under the three investigated scenarios

Scenario	Steps	Final Reward
Base	31	139
High Stochasticity	35	130
Small Discount Factor	500	-500 (agent stuck, maze unsolved)

Comparison of Policy Iteration & Value Iteration

Through my experimental results, I found that both policy iteration and value iteration converge to the same optimal policy under the same algorithmic parameters. For a given scenario, the optimal policy learned by the policy iteration algorithm was the same as the optimal policy learned by the value iteration algorithm. This being said, I did find differences between the two in terms of the number of iterations it took each algorithm to converge under a given scenario, as well as the total computation time. I found that policy iteration required more computation time per iteration but often converged faster than value iteration, while value iteration was slightly simpler to implement than policy iteration and converged slower than policy iteration in most scenarios, as summarized in Table 5.

Table 5: Comparison of policy iteration and value iteration in terms of average computation time

Scenario	**Policy Iteration**	**Value Iteration**
	Time to learn optimal policy (Averaged over 20 independent runs)	
Base	3.94 s	4.35 s
High Stochasticity	2.56 s	3.04 s
Small Discount Factor	0.609 s	0.433 s